VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Webová aplikace pro PIM (Personal Information Management)

Web Application for PIM (Personal Information Management)

2012

Jan Žitník

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jan Žitník**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Webová aplikace pro PIM (Personal Information Management)
Web Application for PIM (Personal Information Management)

Zásady pro vypracování:

Cílem této práce je implementovat webovou aplikaci pro organizaci a správu osobních dat, plánování, zaznamenávání poznámek a organizaci schůzek.
Součástí aplikace je emailový klient a RSS čtečka.

Cíle práce jsou následující:
1. Návrh struktury databáze, vytvoření databázové logiky.
2. Implementace aplikačního serveru s obecným API.
3. Vytvoření RIA klientské aplikace komunikující pomocí JSON se serverem.
4. Implementace IMAP klienta jako modul pro server.
5. Implementace RSS čtečky jako modul pro server.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Nikola Ciprich**

Datum zadání: 18.11.2011
Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

Licence GPL

V Ostravě 16. dubna 2012 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 16. dubna 2012 .....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla.

Abstrakt

Cílem této práce je implementovat webovou aplikaci pro organizaci a správu osobních dat. Aplikace je určena pro vytváření poznámek, plánování a orgranizaci událostí. Součástí aplikace je emailový klient a RSS čtečka implementovaná jako modul. Práce popisuje architekturu, tvorbu webového serveru a uživatelského prostředí ve webovém prohlížeči.

Klíčová slova: Python, JavaScript, PIM, DoJo, PostgreSQL

Abstract

The aim of this thesis is to implement web application for managing personal information. Application is intended for creating notes, planning and time management. Email client and RSS reader is also implemented as a part of this project. Architecture design as well as implementation of both server and client parts is also described.

Keywords: Python, JavaScript, PIM, DoJo, PostgreSQL

Seznam použitých zkratek a symbolů

AJAX	–	Asynchronou JavaScript And XML
API	–	Application Programming Interface
CSS	–	Cascading Style Sheet
DOM	–	Document Object Model
GNU	–	GNU is Not Unix
GPL	–	General Public License
GUI	–	Graphical User Interface
HTML	–	Hyper Text Markup Language
HTTP	–	HyperText Transfer Protocol
IMAP	–	Internet Message Access Protocol
JSON	–	JavaScript Object Notation
LDAP	–	Lightweight Directory Access Protocol
PAM	–	Pluggable Authentication Module
PIM	–	Personal Information Management
RFC	–	Request for Comments
RIA	–	Rich Internet Application
RSS	–	RDF Site Summary
SMTP	–	Simple Mail Transfer Protocol
SQL	–	Structured Query Language
URL	–	Uniform Resource Locator
UTF	–	UCS Transformation Format
WSGI	–	Web Server Gateway Interface
XML	–	eXtended Markup Language
DHTML	–	Dynamic HTML
ISO	–	International Organization for Standardization
DBMS	–	Database management system
JIT	–	Just In Time

Obsah

1 Úvod

Každý se už určitě setkal s problémem cokoliv si zapamatovat, vědět kdy je kde jaká schůzka, nebo zapomněl udělat projekt do školy. V dnešní době jsou papírové organizéry nejen nepraktické, ale špatně se zálohují, nelze v nich vyhledávat a mají další nevýhody. Proto se jeví jako nejlepší východisko úchovávat svá data elektronicky.

Na internetu lze najít spousty hotových řešení, ať už placených, nebo zdarma. Jen málo z nich je ale použitelných. Mnoho takových služeb je poskytováno v cloudu a uživatel neví, kdo má k jeho údajům přistup a co se s nimi na pozadí děje. Cílem této bakalářské práce je vyvinout systém, který je šířen pod otevřenou licencí, je jednoduše rozšiřitelný a má zdokumentované API. Právě otevřenost systému umožňuje tvorbu libovolných klientských částí, případně využití v jiných aplikacích.

První část se zabývá vysvětlením pojmů, teorie a principů, na kterých aplikace zakládá. Je zmíněn programovací jazyk Python, jeho základní popis včetně WSGI rozhraní. Dále je zmíněn Javascript a framework DoJo pro tvorbu uživatelských rozhraní v internetovém prohlížeči.

Druhá část bakalářské práce se zabývá návrhem databáze a architekturou aplikace, podrobné vysvětlení fungování serverové části a komunikací s klientskou aplikací.

Třetí část je věnována podrobnému popisu implementace, popisu vzniklých problémů a jejich řešení. Detailněji ukazuje strukturu programu a využití specifických knihoven a frameworků.

2 Teoretický rozbor

2.1 Proč PIM

Projektů pro PIM existuje spousta, placených i zdarma, ale při bližším prozkoumání žádný nebyl vyhovující. Většina takovýchto projektů je realizována jako centralizovaná online služba, nejčastěji v podobě webové stránky. Uživatelé tak sice mají jednoduchý přístup, ale takovýto způsob ukládání osobních dat není bezpečný, protože nikdy nelze odhadnout, kdo je majitel serveru a co s našimi daty ve skutečnosti provádí. Ukládání firemních a citlivých dat v takových službách je tedy nemožné. Existují sice i open source aplikace, ale kvalita a uživatelská přívětivost pokulhává.

Při výběru frameworků a programovacích jazyků byl kladen důraz na nenáročnost, multiplatformnost pro snadnou přenositelnost napříč operačními systémy a výslednou kvalitu finální aplikace.

Pro tvorbu grafického uživatelského rozhraní se využívá internetového prohlížeče, který je nainstalován na většině počítačů a není tedy nutné instalovat další programy. Z toho plyne snadná použitelnost prakticky kdekoliv. Internetové prohlížeče lze v dnešní době využívat pro tvorbu aplikací, které jsou takřka k nerozeznání od nativní aplikace. Využívá se k tomu JavaScript a AJAX, kdy se načte prázdná stránka a JavaScript dynamicky vytváří GUI prvky, které umisťuje na stránku. Při interakci s uživatelem se stránka nikdy nenačítá znovu (refresh), ale data si stáhne ze serveru ve formátu JSON a případně překreslí určitou část uživatelského rozhraní. Takové aplikace se často označují jako RIA - Rich Internet Application.

Na serverovou část byl zvolen programovací jazyk Python pro svou přehlednost a snadné použití. Skripty jsou pak spouštěny webovým serverem (nejčastěji Apache) pomocí rozhraní WSGI. Python je interpretovaný jazyk (často označován jako skriptovací) a jeho interpreter je dostupný pro většinu operačních systémů.

V závěru je pak shrnut celkový výsledek implementace a architekruty, včetně zhodnocení a plánů do budoucna.

2.2 Funkční požadavky

Nároky na PIM aplikaci byly kladeny tak, aby byly splněny základní potřeby uživatele. Důležitý je čistý objektový návrh tak, aby vývoj mohl v budoucnu neomezeně pokračovat, aby aplikace byla modulární a bylo možné chybějící funkcionalitu doprogramovat.

Mezi základní funkce patří:

- Osobní poznámky/deník - přidávání, úpravy včetně mazání
- Seznamy/úkoly - přidávání, výpisy nesplněných úkolů
- Události a schůzky - plánování, přehledy v kalendáři, týdenní a měsíční pohledy
- Aktuality RSS - stahování RSS novinek
- E-Mail zprávy - příjem a elektornická korespondence
- Adresář - evidence, neomezený počet kontaktů

Úkoly, poznámky a události jsou organizovány pomocí tzv. štítků. Štítek je označení určité položky, obdobně jako kategorie. Každá položka může mít neomezený počet přiřazených štítků, podle kterých lze pak seskupovat výpisy.

2.3 JavaScript a RIA aplikace, DoJo

JavaScript je objektově orientovaný programovací jazyk, využívaný při tvorbě webových stránek. Na rozdíl od serverových programovacích jazyků (například Python) sloužících ke generování kódu samotné stránky, JavaScript běží na straně klienta, tedy v prohlížeči.

JavaScript se používá především pro vytváření interaktivních webových stránek. Příkladem použití mohou být nejrůznější kontroly správného vyplnění formulářů, obrázky měnící se po přejetí myší, rozbalovací menu atd. JavaScript se také často používá k měření statistik návštěvnosti.

Společně s jazykem HTML (informační kostrou stránky) a CSS (formátováním vzhledu stránky) je JavaScript součástí DHTML, souboru technik a postupů zaměřených na zlepšení uživatelského rozhraní a zvýšení prožitku z používání stránek. K tomu JavaScript využívá tzv. DOM, rozhraní umožňující přistupovat k jednotlivým prvkům stránky.

JavaScript vyvinula společnost Netscape v roce 1995 a v roce 1998 byl standardizován organizací ISO. [4]

Dojo Toolkit je kolekcí JavaScriptových komponent, které mají pomoci webovému vývojáři. Základem je dojo.js, který obsahuje kolekci „nezbytných“ API pro nejčastější použití a nabízí celou knihovnu funkcí. Dojo je zcela zadarmo pod duální licencí AFL a BSD licencí.

Druhou částí DoJo toolkitu je GUI knihovna dijit a dojox, která poskytuje prvky pro tvorbu uživatelských rozhraní, využívá tříd a dědičnosti.

2.4 Python, WSGI

Programovací jazyk Python vytvořil Guido van Rossum o Vánocích roku 1989. Při své práci v holandském institutu CWI potřeboval jazyk pro psaní utilit pro distribuovaný operační systém, jenž skupina, ve které pracoval, vyvíjela. Měl v úmyslu vytvořit snadno rozšiřitelný jazyk podobný jazyku ABC a podporující výjimky. V únoru roku 1991 na Usenetu ohlásil první veřejnou verzi Pythonu.

Dnes je Python široce používán na mnoha místech, kde je potřeba přehledný a snadno spravovatelný kód. Používá jej i NASA pro uživatelské rozhraní systému řídícího lety raketoplánů, aplikační server Zope, originální implementace protokolu BitTorrent, balíčkovací systém Portage v distribuci Gentoo, stejně tak instalátor Anaconda pocházející z RedHat Linuxu a mnohé jiné větší či menší programové celky.

Python se vyznačuje poměrně netradiční syntaxí založenou na odsazování. Taktéž může být pro začátečníka matoucí, že Python je silně objektový beztypový jazyk s přístupem podobným spíše Smalltalku nežli "běžným" objektovým jazykům jako je třeba Java.

Python je jazyk interpretovaný a často považovaný za skriptovací, přesto je v něm možno psát i docela rozsáhlé programové celky.

WSGI (Web Server Gateway Interface) definuje jednoduché a univerzální rozhraní mezi webovým serverem a webovou aplikací nebo frameworkem v programovacím jazyce Python. Poslední verze jazyka Python, 3.0, vydaná v prosinci 2008, je již podporována modulem mod_wsgi webového serveru Apache. [1]

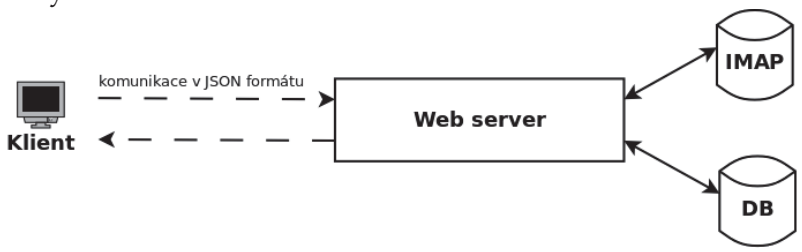
2.5 PostgreSQL

PostgreSQL, často jednoduše Postgres, je objektově-relační databázový systém (ORDBMS). Vydáván je pod licencí typu MIT a tudíž se jedná o free a open source software. Stejně jako v případě mnoha dalších open source programů, PostgreSQL není vlastněn jedinou firmou, ale na jeho vývoji se podílí globální komunita vývojářů a firem.

PostgreSQL je primárně vyvíjen pro Linux resp. pro unixové systémy obecně, nicméně existují i balíčky pro platformu win32. [7]

3 Architektura aplikace

Princip aplikace je rozdělení na dvě části - klientská strana, kterou může zastávat jakákoliv vzdálená aplikace (např. JavaScript aplikace pro dekstop, Java aplikace pro Android) a na serverovou aplikaci. Tyto dvě komponenty spolu komunikují předem pevně daným protokolem ve formátu JSON. To přináší mnoho výhod, např. nezávislost klientské aplikace, kterou si kdokoliv může napsat díky veřejnému API - díky tomu je možné vytvořit alternativní nativní aplikace pro mobilní platformy, případně jiné systémy.



JSON, ve zkratce JavaScript Object Notation je způsob zápisu dat (datový formát) nezávislý na počítačové platformě, určený pro přenos dat, která mohou být organizována v polích nebo agregována objektech. Vstupem je libovolná datová struktura (číslo, řetězec, boolean, objekt nebo z nich složené pole), výstupem je vždy textový řetězec. Složitost hierarchie vstupní proměnné není teoreticky nijak neomezena.

Navzdory názvu, JSON je zcela obecný a může sloužit pro přenos dat (navíc, čitelný pro člověka) v libovolném programovacím nebo skriptovacím jazyce. Data, zapsaná metodou JSON, mohou být samozřejmě uložena a přenášena v souborech; častěji ale přenos probíhá v prostředí internetu (např. s použitím technologie AJAX).

Mezi nedostatky JSON patří to, že neumožňuje definovat znakovou sadu přenášeného obsahu, optimalizace pro přenos binárních dat. Tyto nedostatky platí ale pro některé (slabší) implementace. Nealfabetické znaky v řetězcích a binární data JSON jsou escapovány zpětným lomítkem, za kterým následuje buď jeden z běžně používaných znaků (např. \n pro nový řádek, \t pro tabulátor, \\ pro samotné zpětné lomítko) nebo \u indikující znak z Unicode (UTF-16), za nímž následují čtyři hexadecimální číslice.

Dá se říci, že JSON sází na jednoduchost způsobu uložení dat, srozumitelnost (data jsou čitelná člověkem), platformovou nezávislost a jednotnost (JSON se rychle etabloval) a to vše na úkor velikosti přenášených dat. [5]

3.1 Komunikační protokol

JSON komunikaci zahajuje vždy klientská strana, tedy zjednodušeně řečeno zavolá konkrétní modul na serveru.

Při požadavku na server pomocí URL určíme modul a jeho metodu. Při standardní konfiguraci má skript server/server.wsgi alias v HTTP server na URL /server. Při volání modulu se tedy pomocí server/nazev_modulu/nazev_metody zavolá konkrétní funkce.

Například pro získání dat všech událostí se volá URL server/Events/getData Každý modul by měl implementovat 4 základní metody - getData, update, insert a delete - více informací v kapitole 4.5 4

Požadavek na získání dat je vždy ve tvaru:

```
request:
{
    search: {
        task_start: "2012-04-07T22:30:00"
    },
    limit: 200,
    offset: 200
}
```

Výpis 1: Požadavek

Pomocí asociativního pole search určujeme klíčová slova vyhledávání - např. filtrování událostí podle času. Pomocí klíčových slov limit a offset určujeme úsek dat, které nám server má vrátit - vhodné pro stránkování. Stránkování dat snižuje zátěž serveru a množství přenesených dat, protože málokdy je zapotřebí mít všechna data najednou.

```
response:
{
    data: [
        {col1: 355, col2: ”string”, col3: 128.123, col4: 128, col5: false},
        ...
        ...
    ],
    dataInfo: {
        offset: 200,
        limit: 1000,
    },
    columns: {
        col1: {
            type: ”integer”,
            valid: ”[0−9]+”,
            required: true,
            name: ”Název”,
            visibility: [”edit”, ”static”]
        }
    }
}
```

Výpis 2: Odpověď

Vrácená odpověď má podobně striktní zápis jako požadavek. S příchozími daty je přiložen taky popis jednotlivých sloupců, jejich datové typy, název a omezení. Mezi omezení patří především příznak, jestli je položka při editaci nebo vkládání povinná, případně její validace pomocí regulárního výrazu.

Popis jednotlivých sloupců je pouze doporučený, klientská aplikace se tím nemusí řídit a data si může zobrazovat dle uvážení, kontrola správnosti dat se děje hlavně na serveru, protože nelze spoléhat na data od klienta.

Popis všech parametrů:

- type - datový typ - je pevně svázán s datovými typy serveru, konkrétně knihovna DataTypes
- valid - regulární výraz, podle kterého se data validují
- required - určuje, zda je parametr povinný při editaci nebo vkládání nového záznamu
- name - název sloupce
- visibility - pole, kdy je sloupec/položka viditelná - „edit“ při editaci, „static“ při běžném prohlížení, případně prázdné pole pokud má být položka skryta
- flags - další parametry, které určují povahu sloupce - například „primaryKey“, pokud je sloupec primárním klíčem v databázi

3.2 Návrh databáze

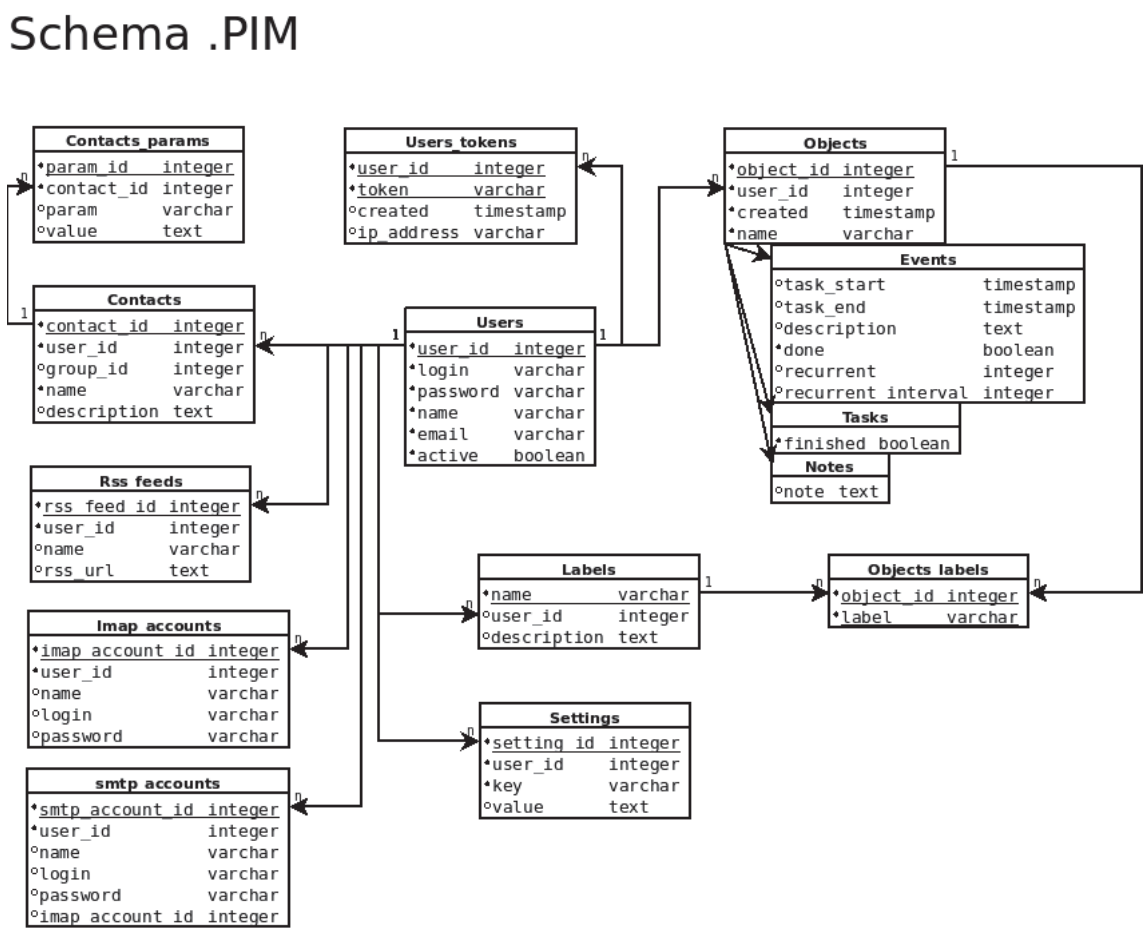
Aplikace využívá databázový systém PostgreSQL, využívá jeho specifické vlastnosti, není tedy jednoduše přenositelná na jiné DBMS.

Využívá se tří základních objektů:

- Note - poznámka
- Event - událost
- Task - úkol

Všechny tyto tři objekty jsou v samostatných tabulkách a dědí z tabulky objects. Dědičnost v PostgreSQL přináší výhodu sjednocených primárních klíčů, případně i ostatních společných sloupců. Sjednocené primární klíče jsou především z důvodu stejných vazebních tabulek na štítky.

Všechny tabulky mají striktní vazbu na Users, což jsou uživatelé systému. Uživatel se musí vždy autentizovat jménem a heslem, při SQL dotazech se pak používá jeho konkrétní ID.



Popis jednotlivých tabulek

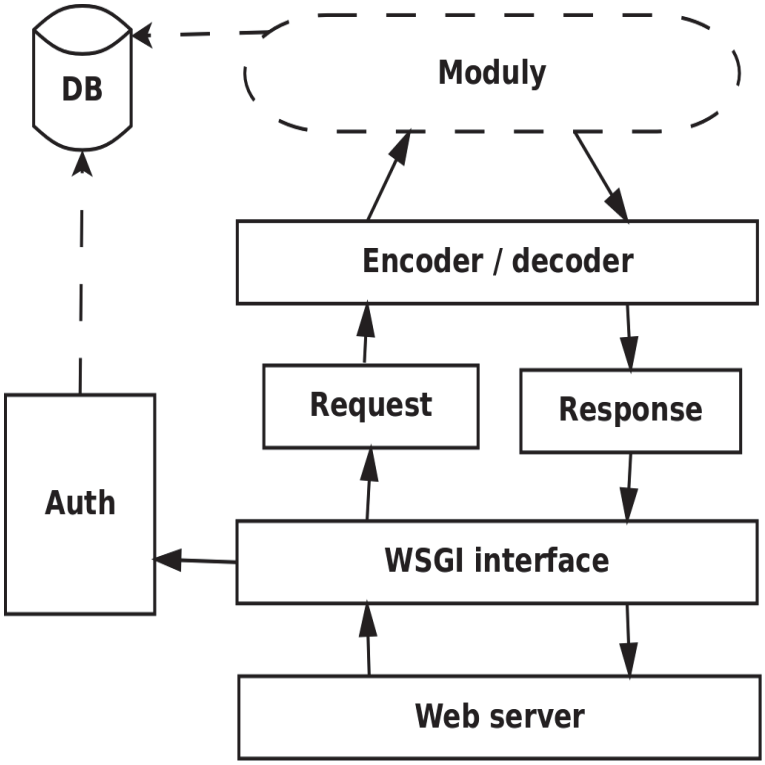
- Users - tabulka pro uživatele aplikace
  - login - přihlašovací jméno
  - password - MD5 hash přihlašovacího hesla
  - name - jméno uživatele
  - email - kontaktní email
  - active - boolean příznak, jestli má uživatel povoleno přihlášení, nebo je blokovaný
- Users_tokens - tabulka pro trvalé přihlášení
  - user_id - uživatel
  - token - vygenerovaný 32 znakový token
  - created - datum vytvoření - tokeny starší 90 dní už nejsou platné
  - ip_address - IP adresa, z které byl token vytvořen
- Contacts - tabulka kontaktů
  - name - název kontaktu
  - description - popis, nebo poznámka
- Contacts_params - jednotlivé položky kontaktu. Každý kontakt může mít libovolný počet položek, například email, telefon a adresu
  - contact_id - cizí klíč s vazbou na kontakty
  - param - název položky - např. email nebo telefon
  - value - konkrétní hodnota - např. konkrétní telefon nebo email
- Rss_feeds - nastavení RSS - jednotlivé účty a jejich URL pro synchronizaci
  - name - název, jen pro přehlednost a výpis v menu
  - rss_url - URL adresa XML feedu
- Imap_accounts - Účty IMAP klienta
  - name - název účtu
  - login - přihlašovací jméno
  - password - heslo - v plaintextu pro přihlášení k IMAPu
  - host - IMAP server
  - port - IMAP port (výchozí 143)
  - ssl - boolean příznak pro zabezpečené spojení SSL

- Smtp_accounts - Účty pro odchozí poštu
  - name - název účtu
  - login - přihlašovací jméno
  - password - heslo - v plaintextu pro přihlášení k IMAPu
  - host - SMTP server
  - port - SMTP port (defaultně 25)
  - ssl - boolean příznak pro zabezpečené spojení SSL
- Objects - rodičovská tabulka ze které dědí Events, Tasks a Notes
  - object_id - sdílený primární klíč, generuje se ze sekvence
  - name - název objektu (Úkolu, události nebo poznámky)
- Events - tabulka událostí v kalendáři
  - task_start - datum začátku události
  - task_end - datum konce události
  - description - textový popis
  - done - boolean příznak, true pokud je událost již dokončena/splněna
  - recurrence - boolean příznak opakování
  - recurrence_interval - interval opakování (počet dní)
- Tasks - tabulka úkolů
  - finished - boolean příznak, true pokud je úkol již splněn
- Notes
  - note - poznámka, využívá se i jednoduchého HTML formátování (odrážky, číslování, styly textu)
- Labels - štítky
  - name - název štítku
  - description - textový popis
- Objects_labels - vazební tabulka štítků na objekty
- Settings - tabulka pro jednotlivá nastavení uživatele
  - key - název konstanty
  - value - hodnota

Dále jsou definovány procedury v jazyce plpgsql, který PostgreSQL poskytuje. SQL procedury jsou spouštěny na SQL serveru, odpadá tedy režie přenosu dat a také je mohou spouštět triggery. Trigger je definovaná činnost, která se má provést při databázové operaci nad tabulkou. Triggery mohou např. spouštět SQL procedury po INSERT, UPDATE nebo DELETE operaci. V PIM aplikaci se využívá trigger ke generování tokenů pro trvalé přihlášení, více v kapitole 4.2. Tento trigger se spustí po vložení záznamu do users_tokens a vygeneruje 32 bitový náhodný řetězec.

Mezi další funcki patří plpgsql procedura pro přiřazování štítků k objektům. Zajistí vytvoření nového štítku, pokud uvedený ještě neexistuje, v opačném případě použije již existující.

4 Implementace serverové aplikace



Serverová strana je napsaná ve skriptovacím jazyce Python, pro spolupráci s webovým serverem využívá WSGI rozhraní, což je nástupce mod_python. WSGI umožňuje, aby aplikace běžela po celou dobu běhu webového serveru a každý příchozí požadavek byl zpracován v jednom vlákně. WSGI požaduje skript obsahující metodu application a předává parametry environ a start_response.

Environ je objekt obsahující proměnné webového serveru, parametry HTTP požadavku a mnoho dalšího.

"start_response" je metoda, která slouží k zahájení odpovědi, předávají se dva parametry, z nichž první je HTTP stavový kód a druhý jsou HTTP hlavičky.

Zjednodušeně vstupní skript serveru může vypadat takto:

```
def application(environ, start_response):

    .... obslužný kód...

    start_response('200 OK', (
        ['content-type', 'text/plain']
    ))
```

Výpis 3: Ukázka WSGI aplikace

Referenční implementace WSGI nijak nepřispívá ke zjednodušení zpracování HTTP požadavku, slouží jen jako mezivrstva HTTP serveru a Python aplikace, pouze předá asociativní pole s proměnnými HTTP serveru, kde se nachází jak tělo požadavku, tak konkrétní URL. V tomto serveru je WSGI skript použit pouze jako mezivrstva mezi webovým serverem a vlastními třídami. WSGI umožňuje tzv. middleware, což je obalení aplikace další vrstvou, v PIM aplikaci se využívá knihovny beaker a middleware SessionMiddleware, která implementuje podporu session. Session dává HTTP serveru možnost uložit si libovolné informace s vazbou na konkrétního uživatele. Používá se především pro identifikaci přihlášeného uživatele a pro zjištění jeho ID (odpovídá user_id v tabulce Users v databázi)

Adresářová struktura serveru má konkrétní význam a rozdělení

- auth - třídy sloužící k autentizaci uživatelů - např. z databáze nebo LDAP
- config - konfigurační soubory
- datasources - třídy zdrojů dat - např. PostgreSQL databáze, souborový systém
- lib - obecné knihovny serveru
  - encoders - vrstvy enkodéru/dekodéru pro serializaci objektů a dat do JSON formátu
- modules - moduly poskytující konkrétní data a operace nad určitými daty
  - mail - IMAP a SMTP klient
    - maillib - knihovna pro přístup k IMAP a SMPT
  - rss - RSS čtečka
  - timetable - návrhář VŠB rozvrhu
- templates - HTML šablony pro přihlášení a samotné web aplikace

4.1 Zpracování HTTP požadavku

Jako základ zpracování HTTP požadavku slouží třída Request, které v konstruktoru předáme environ proměnnou, obsahující vše z HTTP požadavku. Hned při provádění konstruktoru se environ a jeho proměnné rozparsují na interní proměnné get, která obsahuje proměnné z URL (GET požadavek), post, která obsahuje tělo HTTP požadavku (metoda POST) a request, která obsahuje obojí, přičemž při shodě názvu proměnných má přednost POST (POST přepíše GET).

Při příchodu požadavku můžeme také nastavit encoder, tedy jaká knihovna bude použita pro serializaci dat, jako výchozí se používá JSON.

Po vytvoření instance třídy lib.Request se volá metoda dispatch(), která zajišťuje zavolání konkrétního modulu, serializaci dat a vrácení instanci třídy Response.

Třída lib.Response obaluje HTTP odpověď, nastavuje správné HTTP hlavičky a stavové kódy. WSGI při vracení odpovědi přijímá iterátor, třída request rozhraní

iterátoru implementuje, takže lze využít streamování dat po předem nastavené velikosti bloku dat. To vede ke zmenšení zátěže serveru a snížení využité paměti, protože pokud se streamuje velký datový objekt, díky rozhraní iterátoru nemusí být načten v paměti celý, ale pouze předem nastavená velikost bloku.

Environ obsahuje položku "PATH_INFO", která obsahuje část URL za adresou serveru. Tuto část URL rozdělíme podle lomítek, a určíme tak jméno konkrétního modulu. Poslední část URL určuje název metody v modulu.

Pokud k dané URL byl nalezen modul, včetně spouštěné metody, předáme metodě parametry z HTTP požadavku, ale pouze ty, které daná metoda vyžaduje. K tomu slouží knihovna "inspect", konkrétně "inspect.getargspec". Tím je ošetřeno, že metoda dostane pouze ty parametry, které má uvedené ve své definici.

4.2 Autentizace

K jakékoliv interakci se serverem musí být uživatel přihlášen, v opačném případě server vrátí přihlašovací formulář z templates/login.html.

Informace o přihlášeném uživateli včetně jeho user_id se ukládá do session, pokud v session není, automaticky se pokládá za nepřihlášeného. Pro pohodlnost a usnadnění použití může být přihlášení trvalé - od uživatele tedy nebude vyžadováno žádné jméno ani heslo. Trvalé přihlášení funguje na principu ukládání speciálních vygenerovaných tokenů, což jsou náhodné řetězce o délce 32 znaků. Tokeny se ukládají do cookies na straně klienta a do databáze na straně serveru. Cookie je malé úložiště dat v internetovém prohlížeči a tyto data jsou odesílány na server s každým požadavkem. Tímto způsobem se tedy uloží do databáze vygenerovaný token spolu s uživatelským ID a zároveň se odešle ke klientovi. Data, která jsou uložena v cookies, mohou mít nastavenou dobu platnosti. V tomto případě je nastavena platnost na 90 dní. Při příštím spuštění aplikace internetový prohlížeč automaticky odešle na server uložený token, v databázi se ověřuje, kterému konkrétnímu uživateli token patří, ten pak bude považován za přihlášeného. Délka tokenu byla zvolena na 32 znaků z důvodu bezpečnosti - čím delší token, tím složitější je uhádnout token a vydávat se za jiného uživatele. Tokenů lze vygenerovat pro jednoho uživatele více, aby bylo možné využívat trvalé přihlášení z více míst nebo z více prohlížečů.

U nepřihlášeného uživatele server.wsgi vždy pro zpracování HTTP požadavku využívá třídu AuthRequest, která vrací přihlašovací formulář a při odeslání přihlašovacích údajů je ověří pomocí objektu Auth.

Třída Auth implementuje validační funkci "authenticate" přijímající dva parametry - login a heslo, která provádí ověření uživatele v databázi, případně může využívat jiné autentizační mechanismy (např. LDAP, unix PAM atd...).

Metoda authenticate vrací asociativní pole s uživatelským ID a jeho rolí v systému - buďto admin, nebo "user" - což je obyčejný uživatel. Admin má práva přidávat a spravovat další uživatele.

Vzhledem k tomu, že PIM aplikace není předurčena pro široké použití na veřejných serverech, ale spíše na privátních pro pár uživatelů, není povolena registrace, ale uživatele může vytvářet jen administrátor.

4.3 Datové typy, validace dat

V komunikačním protokolu a jeho popisech datových sloupců se uvádí povinný atribut ”type”, což značí datový typ. Datové typy jsou předem definované třídy, které implementují validaci a případně transformaci příchozích dat pro uložení do databáze.

Datové typy jsou deklarovány v lib.dataTypes a povětšinou implementují rozhraní IType. V Pythonu se rozhraní chová stejně jako třída, je to tedy jen ”konvence” pro programátora, nikoliv striktní požadavek na implementaci.

```
class IType(object):

    def __init__(self):
        raise Exception(”Not implemented yet”)

    def validate(self, value):
        raise Exception(”Not implemented yet”)

    def __str__(self):
        raise Exception(”Not implemented yet”)

    def encode(self, value):
        return value

    def decode(self, value):
        return value
```

Výpis 4: Rozhraní datového typu

- __init__, přesněji konstruktor, slouží jen k vlastní inicializaci datového typu. Může přijímat libovolný počet parametrů, které se mu předají při vytváření instance ve specifikaci modulu
- validate je metoda pro ověření správnosti dat, přijímá pouze jednu konkrétní hodnotu v parametru a vrací True nebo False. Samotný kód metody může data ověřovat libovolným způsobem, nejčastěji se využívá knihovny „re“ pro ověření regulárním výrazem, nebo jen např. omezení rozsahu čísla (integer)
- __str__ vrací pouze název datového typu jako řetězec, tedy jak se bude datový typ jmenovat ve specifikaci sloupce v komunikačním protokolu.
- encode ošetřuje příchozí data při ukládání do databáze, nejčastěji se escapují určité sekvence textového řetězce
- decode je inverzní funkce k encode - dekóduje data uložena v databázi při odesílání ke klientovi

Vzhledem k objektovému návrhu datových typů se nové typy dají snadno zdědit z již existujících. Výhodné je to tehdy, pokud chceme například jen informovat klienta, že má

použít jiný prvek (widget) k editaci. Toho se využívá např. u datového typu HTML, což je v podstatě textový řetězec, ale pro editaci se používá HTML editor.

```
class String(IType):
    regexp = None
    def __init__(self, regexp = None):
        self.regexp = regexp

    def validate(self, value):
        if not self.regexp:
            return True

        return False

    def __str__(self):
        return 'String'
```

Výpis 5: Ukázka implementace datového typu

```
class HTML(String):
    def __str__(self):
        return 'HTML'
```

Výpis 6: Dědičnost datového typu

4.4 Automatizované SQL dotazy, transakce

Vzhledem k tomu, že je aplikace založená převážně na SQL databázi, je vhodné manipulaci s databází co nejvíce zjednodušit. Vše se nachází v knihovně lib.sqlReport, která obsahuje třídy pro SELECT, INSERT, UPDATE a DELETE.

Při práci s databází všem třídám (pro jakoukoliv operaci) předáváme definice sloupců. Definice sloupců je instance třídy Column, která obsahuje veškerý popis dat ukládaných ve sloupci, stejně jako se přenáší ke klientovi komunikačním protokolem. Konstruktor této třídy má jediný parametr - asociativní posle s parametry.

- name - název sloupce, bude se zobrazovat na klientské straně
- id - identifikátor, který určuje přesný název sloupce výsledného dotazu, tedy název sloupce, případně název aliasu pokud byl v dotazu použit
- visibility - pole, které určuje viditelnost na klientské aplikaci
  - static - viditelné při zobrazování dat
  - edit - viditelné při editaci

  nebo prázné pole - tedy nebude viditelné nikdy.
- required - restrikce, jestli je hodnota vyžadována při editaci, tedy je povinná
- insertable - boolean příznak, jestli lze data tohoto sloupce vkládat (INSERT) do databáze. Např. primární klíč, který je generován přímo databází má nastaveno false.
- editable - boolean příznak obdobně jako insertable, značí jestli je sloupec upravit pomocí UPDATE
- flags - pole s přídavnými informacemi, např. ”primaryKey”, jestli je sloupec primárním klíčem v databázi
- default - výchozí hodnota sloupce - pokud SQL dotaz vrátí NULL, použije se tato hodnota
- type - instance třídy datového typu, více 4.3
- searchAlias - šablona pro WHERE podmínku. Pokud např. hledáme určitou shodu řetězce pomocí LIKE operátoru, do searchAlias můžeme napsat výraz, kde %s bude nahrazeno konkrétní hodnotou
- value - pevně daná hodnota sloupce

Všechny třídy pro práci s databází dědí ze třídy ColumnParser, která zjednodušuje práci s definicemi sloupců, validaci dat a obsahuje další podpůrné funkce.

4.4.1 SQLReport

Třída SQLReport slouží pro získávání dat z databáze (operace SELECT). Při vytváření instance se v konstruktoru předává zformátovaný řetězec konkrétního SQL dotazu. V SQL dotazu se nesmí uvádět klauzule WHERE, ORDER a LIMIT/OFFSET. Místo těchto klauzulí se zapisují zástupné konstanty. Tyto zástupné konstanty se pak převedou na automaticky vygenerovaný SQL kód. Konstanty se zapisují ve formátu „%(nazev)s“, je to nativní zápis formátování v jazyce Python.

Dostupné konstanty:

- where - místo, kam se má vložit podmínka, která se nastavuje metodou setFilter, případně setFixedFilter
- order - místo, kam se vloží názvy sloupců řazení
- limit - místo, kam se vloží omezení počtu výsledných záznamů, včetně offsetu

Po vytvoření instance třídy SQLReport je nutné předat definice sloupců pomocí metody setColumns. Po předání definice sloupců lze libovolně nastavovat podmínky. K tomu slouží metody setFixedFilter a setFilter. Podmínky předané pomocí setFixedFilter zaručují vynucené použití v každém dotazu a možnost obsáhnout v podmínce i sloupce, které nebyly předem definované. Toho se využívá převážně k vazbě dat na uživatele, jehož ID je v session. Třída SQLReport dále obsahuje metodu setOrder pro řazení výsledků, které se v parametru předává asociativní pole, kde klíče v poli jsou názvy sloupců a hodnoty nabývají „ASC“ pro vzestupné, případně „DESC“ pro sestupné řazení. SQLReport ještě umožňuje omezení počtu výsledků a to metodou setLimit se dvěma argumenty, první pro počáteční pozici v datech a druhý parametr určuje celkový počet výsledků.

```
def getData(self, query):
    result = SQLReport("SELECT * FROM pim.events %(where)s %(order)s %(limit)s")
    result.setColumns(self.columns)
    result.setFixedFilter("user_id = '%s'" % self.session['user_id'])
    result.setFilter(query)
    return result.getReport()
```

Výpis 7: Ukázka použití třídy SQLReport

4.4.2 SQLUpdate

Třída SQLUpdate poskytuje jednoduché rozhraní pro aktualizaci záznamů v databázi. Všechny třídy z knihovny lib.sqlReport se chovají obdobně. Konstruktor SQLUpdate přebírá název tabulky, která se má aktualizovat. Vzhledem k tomu, že UPDATE v SQL databázích vypadá často podobně, stačí jen pomocí metody setKey nastavit WHERE podmínku. Metoda setKey se chová stejně jako setFilter u SQLReport. Nakonec se objektu předají data ve formě asociativního pole metodou setData.

```
def update(self, key, data):
    result = SQLUpdate("pim.events")
    result.setColumns(self.columns)
    result.setKey({
        "event_id": 123
    })
    result.setData(data)

    return result.update()
```

Výpis 8: Ukázka použití třídy SQLUpdate

4.4.3 SQLInsert

Třída SQLInsert zprostředkovává vkládání do databáze. Chování má identické jako třída SQLUpdate, jen se nenastavuje podmínka, která je u operace INSERT zbytečná. Funkce spouštějící SQL dotaz, tedy insert(), vrací primární klíč vloženého záznamu, pokud byl v definicích sloupců uveden.

```
result = SQLInsert("pim.events")
result.setColumns(self.columns)
result.setData(data)
result.setAdditionalData({
    'user_id': self.session['user_id']
})
result.insert()
```

Výpis 9: Ukázka použití třídy SQLInsert

4.4.4 SQLDelete

Poslední třídou, kterou knihovna sqlReport disponuje, je SQLDelete sloužící k mazání záznamů z databáze. Chování má taky obdobné jako třída SQLUpdate, s tím rozdílem, že se nenastavují data, ale pouze podmínka, podle které se řádky odstraňují. Při používání SQLDelete je nutné podmínku uvést, protože PostgreSQL by bez uvedení podmínky smazal všechny záznamy. Z tohoto důvodu třída SQLDelete záměrně

nepodporuje mazání bez podmínky, protože se to v PIM nikde nevyužívá (přinejmenším bývá vždy vazba podmínky na ID uživatele).

```
result = SQLDelete("pim.events")
result.setColumns(self.columns)
result.setFilter({
    'event_id': 123
})

result.delete()
```

Výpis 10: Ukázka použití třídy SQLDelete

4.4.5 SQL rozhraní

Pro připojení k PostgreSQL se na serveru vytváří pool, tedy objekt, který obsahuje blíže nespecifikované množství trvalých připojení k databázi a tyto připojení se z poolu vybírají a vrací. Při práci s databází je z poolu vyžádán objekt připojení, ten se z poolu vyjme po dobu, co se s ním pracuje, pak se do poolu vrátí. Přístup předem vytvořených spojení v poolu má výhodu ve zmenšení režie neustálého připojování a odpojování k databázovému serveru. Každé připojení vyjmuté z poolu se do poolu zase musí vrátit, jinak by vznikl tzv. connection leak, v poolu by objekty připojení došly a aplikace by zhavarovala na nedostatek připojení.

Pool je definován v knihovně datasources.postgres, konkrétně třída ConnectionPool. Tato třída implementuje kromě konstruktoru 3 metody.

- getconn - získává objekt připojení z poolu
- putconn(objekt připojení) - vrací připojení zpět do poolu
- query(sql dotaz) - jednoduché vykonání SQL dotazu, funkce obstará získání připojení, vykonání dotazu a následné vrácení připojení do poolu

Získaný objekt připojení poskytuje metodu cursor(), která vytváří kurzor pro vykonávání SQL dotazů. Kurzor je objekt, který umožňuje vykonání SQL kódu z Pythonu a existuje po celou dobu připojení k databázi. Kurzor vytvořený ve stejném objektu připojení není izolovaný, tedy změny jednoho kurzoru uvidí jiný kurzor stejného objektu, i když nebyl proveden commit

4.4.6 Transakce

Transakce je uspořádaná skupina databázových operací (dotazů, procedur), která se vnímá a provádí jako jediná jednotka a to celá, nebo vůbec ne. Nikdy nesmí nastat případ, kdy se vykoná jen její část. Transakce je vhodné používat při změnách ve více tabulkách, případně řetězení SQL operací. Tyto operace se tak provedou buď to všechny, nebo žádná.

Třída pro vykonávání transakcí SQLTransaction je implementována ve stejné knihovně jako pool připojení, tedy v datasources.postgres. Transakce se provádí tak, že se vytvoří instance této třídy, která poskytuje kurzor. S tímto kurzorem se vykonávají SQL dotazy, při úspěšném provedení všech dotazů se spustí metoda commit() a změny se trvale uloží do databáze.

Třídy pro práci s databází, konkrétně SQLUpdate, SQLInsert, SQLDelete mohou přebírat v konstruktoru instanci SQL transakce. Tímto tyto třídy použijí pro své SQL dotazy stejný objekt připojení. Pak se v kódu pomocí výjimek provede commit, který změny uloží nebo rollback, který vrátí změny do stavu před započetím transakce.

```
transaction = datasources.postgres.SQLTransaction()

try:
    a = SQLDelete(”pim.events”, transaction)
    ...
    b = SQLUpdate(”pim.events”, transaction)
    ...
    b = SQLInsert(”pim.events”, transaction)

    transaction.commit()

except Exception as e:
    transaction.rollback()

finally:
    transaction.close()
```

Výpis 11: Ukázka použití transakcí

4.5 Moduly, rozhraní modulů

Moduly, které mají poskytovat data klientovi, by měly implementovat specifické rozhraní a vracet data ve sjednoceném formátu popsaném v kapitole 3.1. Rozhraní je definováno v knihovně lib.Module, kterou lze zdědit nebo implementovat znovu. Pro rychlost a jednoduchost psaní nových modulů, lib.Module implementuje základní operace s databázemi, které jsou natolik automatizované, že stačí jen definovat sloupce a databázovou tabulku. Z těchto definic se vygenerují patřičné operace SELECT, INSERT, UPDATE a DELETE. Pokud generované operace nejsou vyhovující, lze vždy danou metodu předefinovat.

- __init__(session) - konstruktor, přijímá pouze objekt session, který obsahuje uložené data mezi jednotlivými HTTP požadavky
- getData(query, limit, offset) - metoda, která vrací konkrétní data. Objekt query slouží k vyhledávání, limit a offset ke stránkování dat
- getColumns - metoda bez parametrů, která vrací pouze specifikaci sloupců, více v kapitole 4.4
- update(key, data) - metoda pro aktualizaci dat. Objekt key určuje konkrétní podmínku, které řádky se mají aktualizovat. Data obsahují nové hodnoty
- insert(data) - metoda pro vkládání dat
- delete(key) - metoda pro mazání dat. Objekt key obsahuje podmínku, které řádky se mají smazat

Třída lib.Module vyžaduje, aby byly v třídní proměnné „columns“ definovány sloupce. Definice sloupců jsou instance třídy lib.sqlReport.Column v poli. Dále také požaduje třídní proměnnou „table“, která specifikuje tabulku, se kterou se bude pracovat. Automaticky vygenerované SQL dotazy vždy ošetřují příchozí data a kontrolují jejich správnost podle datových typů.

Při předefinování metod lze vykonat více SQL operací a především je vhodné všechny tyto operace provádět v transakcích.

4.6 Modul RSS čtečky

Součástí PIM aplikace je základní implementace RSS čtečky. V databázi k tomuto účelu existuje tabulka rss_feeds, kde má každý uživatel uložené URL k jednotlivým RSS feedům.

Modul je rozčleněn na 3 třídy:

- Feed - výpis všech záznamů z konkrétní URL
- FeedList - seznam dostupných RSS feedů
- rssSettings - administrace RSS feedů

K samotnému parsování RSS feedu je využívána Python knihovna feedparser. Ta poskytuje třídu feedparser.parse, která vrací objekt s RSS položkami v atributu „entries“. Ty se pak pouze zformátují tak, aby odpovídaly specifikacím sloupců a odešlou na klienta k zobrazení.

4.7 Modul E-Mailového klienta

Základem pro emailový klient je využití vestavěné knihovny imaplib a smtplib v Pythonu. Imaplib poskytuje nízkoúrovňové rozhraní pro IMAP účty, práce s tímto rozhraním je těžkopádná a nepohodlná. To vedlo k implementaci vrstvy využívající imaplib a implementaci funkcí usnadňující tvorbu modulu.

Modul emailového klienta je rozdělen na tyto části:

- folderTree - pouze vrací strom složek
- imapSettings - poskytuje IMAP nastavení
- smtpSettings - poskytuje SMTP nastavení
- mailList - vrací seznam emailů konkrétní složky
- mailMessage - vrací tělo emailu a umožňuje odeslat nový email

IMAP (Internet Message Access Protocol) je internetový protokol pro vzdálený přístup k e-mailové schránce. Na rozdíl od protokolu POP3 umí IMAP pracovat v tzv. on-line i off-line režimu a nabízí pokročilé možnosti vzdálené správy (práce se složkami, přesouvání zpráv, prohledávání na straně serveru a podobně). V současné době se používá protokol IMAP4 (IMAP version 4 revision 1 - IMAP4rev1), který je definován v RFC 3501. [8]

Vrstva pro IMAP obsahuje pouze jednu třídu maillib.Imap. Ta v konstruktoru přebírá ID uživatele a ID účtu, ke kterému se má připojit. Všechny IMAP účty včetně nastavení jsou uloženy v databázové tabulce imap_accounts. ID uživatele IMAP vrstva přebírá pouze pro kontrolu, jestli konrétní účet tomuto uživateli náleží. Předejde se tak útoku, kdy by uživatel zkoušel v požadavku uvádět cizí ID. IMAP vrstva si vytvoří instanci IMAP4 objektu z imaplib, případně IMAP4_SSL, pokud jde o zabezpečené spojení.

Vrstva poskytuje především konverzi IMAP struktur do vlastních formátů, například parsování adresářové struktury do stromu a převod datumu na nativní objekty jazyka Python. Dále tato třída poskytuje metody:

- login(login, password) - přihlášení uživatele
- getFolderTree - vrátí adresářový strom v asociativním poli
- getMailList(directory, searchParams) - vrátí seznam emailů konkrétní složky (výchozí je „INBOX“)

- getMail(mailId, folder) - vrátí tělo konkrétního emailu. Je nutné předat i název složky
- close - korektní ukončení spojení

Pro odesílání se využívá vestavěné knihovny smtplib. Její použití je snadnější a jednodušší než imaplib, takže bylo zbytečné vytvářet další vrstvu. V modulu se vytvoří spojení na SMTP server včetně přihlášení, přidá se emailová zpráva v podobě MIME a odešle.

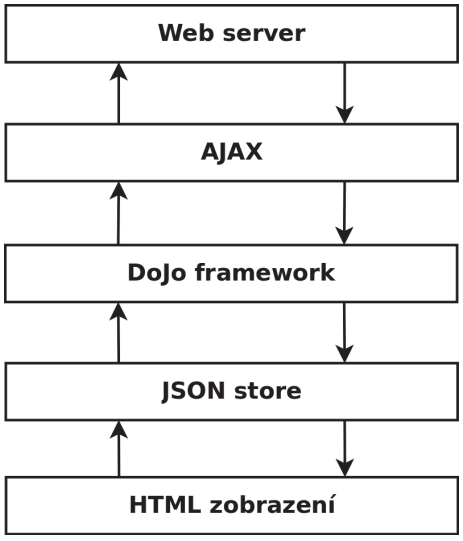
MIME, plným názvem Multipurpose Internet Mail Extensions („Víceúčelová rozšíření internetové pošty“), je internetový standard, který umožňuje pomocí elektronické pošty zasílat zprávy obsahující text s diakritikou, lze k ní přiložit přílohu v nejrůznějších formátech, umožňuje funkci digitálního podpisu apod. V současné době ho využívají i další protokoly aplikace (např. HTTP). Standard MIME je definován šesti dokumenty: RFC 2045, RFC 2046, RFC 2047, RFC 4288, RFC 4289 a RFC 2049. [9]

Samotná zpráva je pak včetně HTML formátování zabalena ve třídě MIMEText s kódováním UTF-8.

5 Implementace klientské aplikace

Aby byla klientská aplikace multiplatformní a dostupná kdekoliv, využívá se RIA architektury v internetovém prohlížeči. Celá klientská aplikace funguje na principu načtení počáteční prázdné stránky bez jakýchkoliv elementů a následné vykreslování (renderování) jednotlivých GUI prvků pomocí programovacího jazyku JavaScript. V dnešní době je rychlost JavaScriptu více než dostačující, využívá se mnoho optimalizací včetně JIT compileru.

Aplikace vykreslí kompletní GUI pomocí vlastních prvků a veškeré data odesílá i stahuje komunikačním protokolem pomocí AJAX požadavku. AJAX je asynchronní požadavek na server, který běží na pozadí a při odezvě vyvolá předem definovanou událost. Veškerá interakce s uživatele využívá AJAX požadavků a překreslení pouze určité oblasti aplikace, stránka se tedy nikdy nepřekresluje celá, tímto se docílí téměř okamžité odezvy bez problikávání obdobně jako v nativních aplikacích.



5.1 DoJo - objektový JavaScript

Vykreslování jednotlivých GUI komponent a widgetů usnadňuje použitý framework DoJo, který do JavaScriptu zavádí objektivně orientované programování, včetně dědičnosti tříd a funkce pro práci s objekty.

DoJo obsahuje nebo je rozděleno na 3 části. Základní část je nedělitelná, často označována jako „core“ obsahuje funkci pro práci s DOM objektem. DOM objekt je strom rozparsované HTML stránky, tedy rozhraní mezi HTML strukturou a JavaScriptem. Jakákoliv změna v DOM stromu se ihned projeví v HTML struktuře a naopak. Základní část DoJo toolkitu taktéž obsahuje podpůrné součásti pro objektově orientované programování, deklaraci tříd a dědičnosti, včetně ostatních nástrojů např. pro AJAX požadavky na server, snadné CSS stylování HTML prvků a mnoho dalšího.

Druhou částí je knihovna Dijit, která staví na objektovém programování a vytváří snadno použitelné GUI prvky, kontejnery pro tvorbu GUI, ovládací a vstupní widgety.

Třetí částí je knihovna DojoX, označována jako DoJo Experimental, kde se nachází experimentální, nové, případně neotestované části jak GUI prvků, tak samotných funkcí DoJo. Z DojoX se jednotlivé části přesunují do „core“ nebo dijit po otestování a prohlášení za stabilní.

5.1.1 GUI prvky

Veškeré GUI prvky dědí ze základní třídy dijit._Widget. Tato třída implementuje základní strukturu GUI prvku tak, jak ji vyžaduje DoJo. Základní, kořenový HTML prvek, se nachází v třídní proměnné domNode. Tento HTML prvek může obsahovat libovolnou HTML strukturu, ale ve frameworku se pracuje pouze s celou třídou, do HTML prvků by se zasahovat nemělo.

dijit.Widget implementuje základní události, které můžeme přepsat, nebo využít návrhového vzoru observer a na tyto události se napojit. Životní cyklus DoJo widgetu:

- constructor - konstruktor objektu
- postMixInProperties - metoda, která volána před samotným započetím vykreslování, tady lze ovlivnit všechny počáteční proměnné objektu
- buildRendering - pokud je zděděn i předek dijit._Templated, zde se parsuje HTML šablona do DOM stromu
- postCreate - typicky metoda pro vykonání vlastního kódu po vytvoření prvku
- startup - metoda, která je volána po vložení prvku do DOM stromu, tedy do HTML stránky
- destroy - destruktor, v této metodě by se měly odstranit veškeré interní objekty a instance pro garbage collector

```
dojo.declare(”myObject”, dijit._Widget, {
    constructor: function(args) {
        this.param1 = args.param1;
        this.param2 = args.param2;

        this.inherited(arguments);
    },

    onClick: function(event) {
        console.log(”clicked”, event);
    }
})
```

Výpis 12: Třídy v DoJo toolkit

Při vytváření tříd pomocí DoJo je vhodné dbát na předem určenou štábní kulturu. Především při vytváření instance se parametry vždy předávají jako asociativní pole (v JavaScriptu a dále jako objekt). Tímto objektem lze při vytváření instance přepisovat i jednotlivé metody.

```
var instance = new myObject({
    name: ”nazev”,
    onClick: dojo.hitch(this, function(event) {
        this.onMouseClick(event);
    })
});
```

Výpis 13: Třídy v DoJo toolkit

Vzhledem ke koncepci JavaScriptu je kontext volané funkce vždy kontext původní, tedy nepřepsané funkce. Pokud třída „myObject“ definuje funkci onClick, a v konstruktoru je přepsána jinou funkcí, kontext (třídní proměnná this) je třída myObject, nikoliv třída ve které je instance vytvořena a funkce přepsána. Předejít této situaci se dá pomoci funkce dojo.hitch, která obalí funkci do správného kontextu, v tomto případě třídy ve které vytváříme instanci, nebo pomocí uzávěrů, což se ale příliš nedoporučuje z důvodu možných vzniků memory leaků (úniku paměti).

5.2 Architektura a design aplikace

V klientské aplikaci se využívá především zobrazovacích prvků z knihovny Dijit, případně dědění ze základních tříd a implementace vlastních prvků.

Aplikace se dělí na dvě hlavní části - GUI a Widgety. Widgety implementují jednotlivé zobrazovací, případně vstupní prvky a GUI pouze zobrazuje tyto prvky na obrazovce, případně umisťuje do konkrétních kontejnerů. Tomu odpovídá i adresářová struktura aplikace:

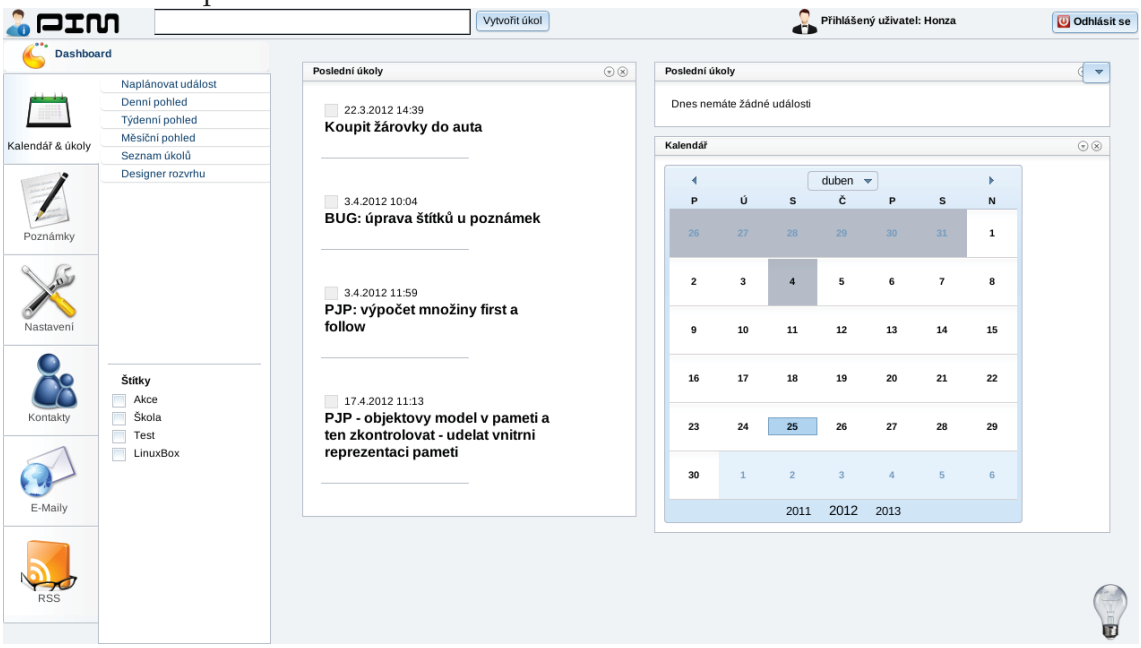
- app
  - config - statické konfigurační soubory
    - Dashboard.js - dostupné widgety pro hlavní obrazovku
    - SiteMap.js - konfigurace jednotlivých pohledů a menu
  - gui - implementace GUI
    - init.js - inicializace a spouštění aplikace
    - Page.js - implementace zobrazené stránky (menu, kontejnery, vykreslování)
  - widgets - vlastní zobrazovací a vstupní widgety
  - templates - šablony pro DataForm, více v kapitole 5.4.1

Inicializace aplikace se provádí ve scriptu app/init.js, který vytváří instanci třídy app.gui.Page, tedy hlavní stránky, a napojuje se na základní události jako je hashChange a document.onkeypress. Událost hashChange je vyvolána při změně URL za znakem "#", která se mění při změně sekce v menu - tím se může přímo pomocí URL určit obsah k zobrazení, takže lze odkazem otevřít podstránku i v JavaScriptu. Document.onkeypress je událost vyvolaná po stisku klávesy, takže lze využívat předem definované klávesové zkratky.

Dále se při inicializaci vkládá instance Page.js do DOM stromu, tím se zobrazí na stránce a zavolá startup(), která se volá na každý widget po vložení do DOM stromu.

Vzhledem k možnostem JavaScriptu init.js rozšiřuje některé nativní objekty, jako je String a Date pro snadnější práci s datem.

Základní koncept uživatelského rozhraní:



5.2.1 Vykreslování stránek

Třída app.gui.Page vytváří základní strukturu stránky - což je hlavička, kde se zobrazuje panel o přihlášeném uživateli, levé menu a kontejner pro vykreslování obsahu jednotlivých pohledů. Po celou dobu běhu aplikace se app.gui.Page nemění, pouze se překresluje kontejner s pohledem konkrétní sekce.

Překreslování obsahu obsluhuje metoda onHashChanged, která je napojena na nativní událost prohlížeče hashChange reagující na změnu URL. Metoda nejprve odstraní původní obsah, zajistí korektní odstranění původní instance a vytvoří novou instanci odpovídajícího obsahu. Instance obsahu a jejich vazba na URL je konfigurovatelná v konfigurační souboru app.config.SiteMap, podle kterého se i vykresluje menu.

```
app.config.SiteMap = {
    mails: {
        title: "E−Maily",
        order: 3,
        menuProvider: app.gui.mail.MenuProvider,
        items: {
            mailBrowser: {
                page: app.gui.mail.MailBrowser,
                title: "Prohlížení",
            },
            newMail: {
                page: app.gui.mail.NewMail,
                title: "Napsat E−Mail",
            },
        }
    },
}
```

Výpis 14: Struktura SiteMap

SiteMap konfigurace je JavaScriptový objekt a jeho položky se větví do stromu. První úroveň vytváří kategorii v menu a druhou úroveň, tedy položky v kategorii vytváří položka "items". Klíč názvu objektu je unikátní identifikátor, který se používá i v URL. Položky v kategorii se vykreslují automaticky dle konfigurace, nebo lze definovat vlastní widget uvedený v menuProvider. Tento widget se vykreslí v kategorii místo ostatních položek (na konfiguraci položek se pak nebere ohled).

Po překreslení obsahu se na instanci podstránky dále volá metoda onHashChanged, aby změna URL "prosákla" níže. Pokud se nezmění v URL identifikátor obsahu, pouze se událost onHashChanged zavolá na samotný objekt obsahu.

5.3 Komunikace se serverem

Komunikace probíhá převážně striktně definovaným protokolem, více v kapitole 3.1. Konkrétní způsob komunikace může být odlišný pro jednotlivé moduly, definice komunikačního protokolu je především doporučená, ale ne striktně vyžadována.

JavaScript umožňuje asynchronní HTTP požadavky na server, takže při interakci s uživatelem se mohou odesílat/přijímat data bez vlivu na odezvu GUI. K tomu framework DoJo poskytuje API dojo.xhr, v tomto případě se používá funkce dojo.xhrPost, která odesílá data metodou POST.

5.3.1 JsonStore

Všechny widgety frameworku DoJo preferují oddělení datové vrstvy od prezenční, každý widget poskytující rozhraní nad nějakými daty vyžaduje objekt Store, který představuje datový model. DoJo sice disponuje škálou Store tříd, ale vzhledem k definici vlastního komunikačního protokolu je nutné implementovat vlastní Store třídu. DoJo poskytuje rozhraní k implementaci těchto tříd, které se rozdělují na 3 základní typy:

- dojo.data.api.Read - rozhraní potřebné ke čtení dat
- dojo.data.api.Write - rozhraní potřebné k ukládání dat (vkládání, editaci, mazání)
- dojo.data.api.Identity - rozhraní určené k manipulaci s primárními klíči

Store třída v PIM klientské aplikaci, app.JsonStore, implementuje všechny tři tyto rozhraní. Splňuje tak všechny předpoklady k plnohodnotnému využití všech dostupných i vlastních widgetů pro operaci s daty.

V konstruktoru JsonStore je jeden povinný argument ”module“, a to název modulu na serveru, se kterým bude probíhat veškerá komunikace.

```
this.store = new app.JsonStore({
    module: ”Tasks”
});

this.form = new app.widgets.form.DataForm({
    store: this.store,
    query: {
        finished: false
    }
});
```

Výpis 15: Příklad užití JsonStore

Pro získání dat ze Store slouží metoda fetch, samotná třída už pak zajistí, jestli data bude stahovat ze serveru, případně načte ze své cache. Vše záleží na interní implementaci třídy. Metoda fetch přijímá následující argumenty (tak jako všechny metody z DoJo framworku - v asociativním poli)

- query - objekt pro vyhledávání dat,
- onBegin - událost vyvolaná při začátku načítání dat
- onItem - událost vyvolaná při zpracování každého řádku
- onComplete - událost vyvolaná po načtení všech dat
- onError - událost vyvolaná při chybě
- scope - kontext událostí
- start - začátek offsetu dat
- count - maximální počet řádků
- sort - objekt s argumenty pro třízení dat

Před samotným stažením dat si každá instance JsonStore načte ze serveru specifikace jednotlivých sloupců, podle kterých pak formátuje argumenty pro vyhledávání dat. Všechny filtry uvedené v query musí striktně odpovídat komunikačnímu protokolu. Každý datový typ definuje metodu encode() a decode(), které formátují data do odpovídajícího formátu.

Pro vložení nového záznamu do JsonStore slouží metoda newItem přijímající objekt nového záznamu jako parametr. Pro úpravu konkrétního atributu JsonStore poskytuje metodu setValue přijímající 3 parametry:

- item - záznam, který bude upraven
- attribute - název atributu
- value - nová hodnota

Ke smazání záznamu existuje metoda deleteItem() s jediným parametrem - záznam, který má být smazán.

Všechny změny (vložení, úprava, smazání) se ukládají pouze lokálně do konkrétní instance JsonStore. K odeslání ke zpracování na server je nutné zavolat metodu save(), která najednou odešle veškeré změny v instanci.

5.4 Widgety

Widgety jsou základní stavební prvky celého uživatelského rozhraní. Widget si lze představit jako konkrétní část uživatelského rozhraní, která má jednoznačené určení. V PIM aplikaci se kromě widgetů z DoJo frameworku využívají často nově implementované widgety pro konkrétní použití (např. kalendáře, formuláře atd...)

Mezi nejdůležítější widgety patří:

- LabelSelector - výběr štítků při editaci
- Menu - sestavení kompletního menu podle konfigurace v SiteMap
- Notify - pop-up oznamování důležitých událostí (příjem emailu atd...)
- ContactDetail - pohled na detail kontaktu
- DateTimeTextBox - vstupní prvek pro zadávání data a času
- MonthView - měsíční pohled na kalendář včetně rozvržení úkolů
- Overview - plocha s widgety ve sloupcích - používá se pro dashboard
- DataForm - komplexní widget pro zobrazování a úpravu dat
- MailMessage - detailní pohled na emailovou zprávu
- WeekCalendar - týdenní pohled na kalendář

5.4.1 DataForm

DataForm widget patří mezi nejrozsáhlejší widgety v aplikaci. Zajišťuje zobrazování, editaci a vkládání jakýchkoliv dat. Jako úložiště dat používá instanci DoJo Store, v tomto případě povětšinou JsonStore. DataForm umožňuje zobrazit data i jejich editaci v libovolné podobě, včetně neomezeného rozmisťování jednotlivých vstupních prvků. Způsobem zobrazování je DataForm tak univerzální, že jej lze použít na zobrazení všech dynamicky načítaných dat.

Základem pro zobrazování je HTML šablona, která vytváří rozvržení jednotlivých vstupních, případně zobrazovacích prvků. Framework DoJo disponuje třídou dijit._TemplatedMixin, která parsuje HTML šablony do widgetu a vytváří jednoduchý přístup ke konkrétním HTML prvkům. Každý HTML prvek může mít atribut dojoAttachPoint (nově data-dojo-attach-point, který je HTML 5 kompatibilní). Tento atribut, resp. jeho hodnota, určuje třídní proměnnou, která bude sloužit pro přístup k tomuto prvku. Např: <**div dojoAttachPoint**="prvek1"> </**div**> bude v deklarované třídě přístupný jako this.prvek1. Celá HTML šablona musí být obalena v jednom kořenovém prvku , ten pak bude dostupný jako domNode ve widgetu, viz 5.1.1.

DataForm rozparsuje HTML šablonu a na určené místa vloží vstupní nebo zobrazovací GUI prvky podle datového typu. Místa, na která se tyto prvky vkládají, musí mít striktně danou hodnotu atributu dojoAttachPoint. Hodnota dojoAttachPoint musí obsahovat vždy identifikátor sloupce („id“ ve specifikaci, viz 3.1) a „_field“ nebo „_label“.

- _field označuje prvek, kam se má vložit vstupní nebo zobrazovací widget podle datového typu
- _label označuje prvek, kam se má vložit název (popisek) vstupního pole - ten odpovídá poli „name“ ve specifikaci sloupce

Jedinou výjimkou je prvek pro vložení tlačítek pro editaci a smazání. U takového prvku musí atribut dojoAttachPoint nabývat hodnoty „buttonNode“. V HTML šablonách lze využívat veškeré dostupné HTML formátování včetně CSS stylování. Šablony pro DataForm jsou uloženy v adresáři app/templates rozdělené podle názvů modulů (název šablony musí odpovídat přesně názvu modulu).

DataForm má dva základní stavy - static a edit, přičemž static pouze zobrazuje data v needitovatelné podobě, zatímco edit obsahuje vstupní widgety pro úpravu dat. Přepínání těchto stavů se provádí pomocí metod startEdit() a stopEdit(), případně se konstruktoru předá parametr insert: true pro vložení nového záznamu.

Veškeré převody dat ze serveru na požadovaný formát, případně vykreslení v konkrétním vstupním prvku, se provádí automaticky podle datových typů, stejně jako validace při ukládání.

```
<div>
 <table style="width: 100%">
  <tr>
   <td dojoAttachPoint="param_field" style="width: 30%"></td>
   <td dojoAttachPoint="value_field" style="width: 30%"></td>
   <td class="inlineButtons"><div dojoAttachPoint="buttonNode"></div></td>
  </tr>
 </table>
</div>
```

Výpis 16: Příklad definice šablony

```
this.paramsStore = new app.JsonStore({
    module: "ContactsParams"
});

this.paramsForm = new app.widgets.form.DataForm({
    query: {
        contact_id: this.contact_id
    },
    store: this.paramsStore,
    staticData: {
        contact_id: this.contact_id
    }
});
```

Výpis 17: Příklad použití DataForm

Pro samotné vykreslení šablony se v DataFormu používá třída Field, která dědí z třídy _TemplatedMixin. DataForm dokáže těchto vykreslovacích prvků zobrazit neomezený počet pod/za sebou, takže lze vykreslovat i dlouhé seznamy dat. Každý tento prvek se pak chová nezávisle na sobě - lze editovat a mazat.

DataForm akceptuje následující parametry:

- store - instance DoJo Store třídy
- query - filtrování dat
- offset - pole s limitem a offsetem dat
- sort - třídění dat podle sloupců
- insert - boolean příznak, který určuje, jestli se budou vkládat nové data
- visibleButtons - pole s viditelnými tlačítky. Může nabývat hodnot: ["edit", "delete", "cancelEdit"]
- staticData - data, která jsou předem definována a nelze je během editace změnit. Jsou také odeslána na server

5.4.2 Dashboard

DashBoard je systém pro umisťování malých widgetů do sloupců na obrazovce. Používá se především pro ucelený přehled nad všemi daty, například počet nepřečtěných emailů, poslední RSS nebo následující úkoly. Základem tohoto widgetu je třída dojox.layout.GridContainer, která implementuje rozmisťování potomků do sloupců.

DashBoard je jen wrapper nad třídou dojox.layout.GridContainer, který zavádí především konfiguraci této plochy. Veškeré nastavení provedené uživatelem se ukládá na serveru, při příštím načtení se tato konfigurace načte a nastaví GridContainer. Ve výchozím nastavení je DashBoard prázdný a obsahuje 3 sloupce pro umisťování widgetů. Widget pro dashboard je jednoduchý panel libovolné velikosti implementující rozhraní dojox.widget.Portlet. Toto rozhraní vytváří kolem samotného widgetu titulní pruh s možností přesunu mezi DashBoard sloupci a tlačítkem pro zavření / skrytí widgetu.

Do nastavení DashBoard také spadá dialogové okno pro výběr widgetu a přidání na plochu. Dostupné widgety jsou staticky uloženy v konfiguračním souboru app.config.DashBoard.

```
app.config.Dashboard = {
    availableWidgets: [
        {
            name: ”INBOX”,
            description: ”Počet nepřečtených emailů”,
            widget: app.widgets.dashboard.Inbox

        }
    ]
}
```

Výpis 18: Konfigurace widgetu v DashBoard

Pro ukládání konfigurace včetně rozmístění jednotlivých widgetů na ploše existuje v databázi tabulka settings. Konfigurace DashBoardu se ukládá jako JSON, protože slouží jen pro JavaScriptového klienta.

```
{
    columns: 3,
    widgets: {
        ”0”: [
            ”app.widgets.dashboard.Tasks”,
            ”app.widgets.dashboard.Events”
        ]
        ”1”: [
            ”app.widgets.dashboard.Inbox”
        ]
    }
}
```

Výpis 19: Konfigurace DashBoardu

5.4.3 RSS čtečka

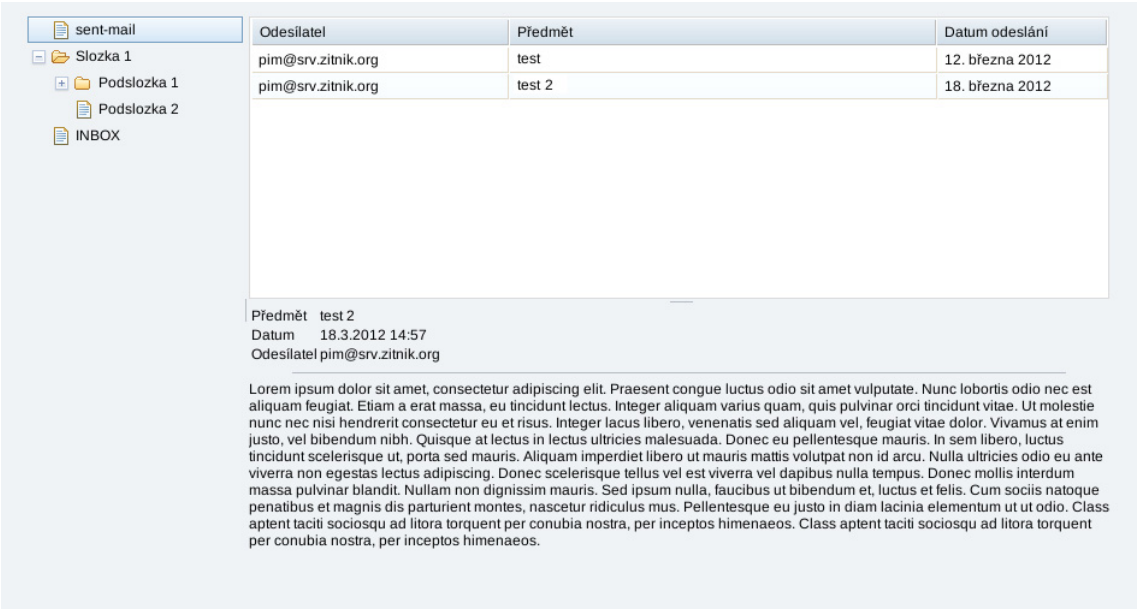
Modul RSS čtečky pouze zobrazuje přijaté data ze serveru ve standardním komunikačním protokolu. Stahování RSS feedu tak probíhá přes server, který data i patřičně rozparsuje na JSON formát, více v kapitole 4.6

Klientská strana se skládá z menu widgetu, který zobrazuje dostupné RSS feedy a implementace DataForm widgetu poskytující zobrazení jednotlivých RSS záznamů. Menu widget stahuje konfiguraci RSS účtů ze serveru, která je uložena v databázové tabulce rss_feeds (vázána na uživatele).

Samotné prohlížení RSS využívá DataForm a předdefinovanou HTML šablonu. Vzhledem k automatizovanému zobrazovaní DataFormu stačí pouze předat instanci JsonStore, který komunikuje s RSS modulem na serveru, všechny RSS se pak vykreslí dle struktury v HTML šabloně.

5.4.4 E-Mailový klient

Emailový klient umožnuje odesílat a přijímat emaily z libovolného IMAP/SMTP účtu. Modul obsahuje vlastní Menu widget, který slouží k zobrazení jednotlivých účtů - aplikace má podporu pro libovolné množství IMAP i SMTP účtů. Každý IMAP účet odpovídá jedne položce v Menu widgetu.



Pro každý účet je zobrazen strom složek, využívající komponentu dijit.Tree. Komponenta dijit.Tree vyžaduje datový modelu, který je implementován ve třídě app.ForestStoreModel. Tento datový model zajišťuje dynamické načítání jednotlivých větví stromu ze serveru, takže strom nemusí být načtěn v paměti celý. Model také poskytuje informace o tom, která položka ve stromu má další potomky. Toho využívá dijit.Tree k vykreslení rozbalovací struktury.

Seznam emailů je vázán na vybranou složku ve stromu. Ve výchozím stavu se automaticky zobrazuje „INBOX“. Zobrazovací prvek pro výpis emailů je dojox.grid.DataGrid, který představuje tabulku s předem defonovanými sloupci. Jako všechny DoJo widgety pracující s daty vyžaduje instanci DoJo store třídy - v tomto případě zase JsonStore.

Zobrazování samotné emailové zprávy zprostředkovává instance třídy DataForm, která si stáhne kompletní rozparsovanou zprávu pomocí JsonStore a pouze vykreslí HTML šablonu.

Odesílání emailových zpráv funguje obdobně. Třída app.gui.mail.NewMail vykresluje základní uživatelské rozhraní pro vytváření zpráv. Zobrazuje 3 vstupní prvky - přijemce, předmět a HTML editor pro samotnou zprávu. Vstupní prvek příjemce je dijit.form.ComboBox, který funguje jako našeptávač. Tento našeptávač pomocí JsonStore komunikuje se serverovým modulem Contacts, který prohledává kontakty uživatele a případně automaticky nabízí a doplňuje emailové adresy do vstupního prvku.

Při odesílání emailové zprávy se volá na serveru metoda sendMail() z modulu mail/MailMessage, která využívá jednoduchý SMTP přístup popsaný v kapitole 4.7

6 Závěr

Výsledkem této bakalářské práce je plně funkční systém splňující základní požadavky na správu osobních informací. Aplikace je rozdělena na dvě nezávislé části - server a klient, což umožňuje implementaci dalších klienstkých aplikací pro různé platformy a použití.

Pro úspěšnou realizaci této práce byl zapotřebí především dobrý návrh a architektura celého systému. Dále také byla nutná dobrá znalost použitých technologií, především programovacích jazyků Python a JavaScript včetně frameworku DoJo.

Realizace byla pro mne přínosná v podobě nově získaných znalostí a problematiky univerzální komunikace mezi klientem a serverem, zpracování informací a tvorbě uživatelských rozhraní v internetovém prohlížeči.

Dokončením bakalářské práce vývoj PIM aplikace nekončí, bude se nadále rozvíjet a především bude snaha o rozšíření uživatelské komunity. Do budoucna je v plánu dále rozšiřovat a vyvíjet obě části, především serverovou stranu pro sdílení informací mezi více uživateli a aplikace pro systém Android. To by mělo výrazně přispět k rozšíření tohoto systému mezi uživatele mobilních telefonů.

Jan Žitník

7 Reference

[1] Švec, Jan, *Seriál Python - Létajíc cirkus* Root.cz : informace nejen ze světa Linuxu [online]. Verze 1.0. 2001 [cit. 2011-04-23]. Létající cirkus. Dostupné z WWW: http://www.root.cz/clanky/letajici-cirkus/

[2] Harms, Daryl - McDonald, Kenneth *Začínáme programovat v jazyce Python* Praha : Computer Press 2008.

[3] Zakas, Nicholas Z., *JavaScript pro webové vývojáře* Praha : Computer Press 2009

[4] *Adaptic* Co je to JavaScript [online]. 2012. Dostupné z URL: <http://www.adaptic.cz/znalosti/slovnicek/javascript>

[5] *Wikipedie - Otevřená encyklopedie* JavaScript Object Notation [online]. 2012. Dostupné z URL: <http://cs.wikipedia.org/wiki/JavaScript_Object_Notation>

[6] Bruce, Momjian *PostgreSQL - praktický průvodce* Praha : Computer Press 2003

[7] *Wikipedie - Otevřená encyklopedie* PostgreSQL [online]. 2012. Dostupné z URL: <http://cs.wikipedia.org/wiki/PostgreSQL>

[8] *Wikipedie - Otevřená encyklopedie* IMAP [online]. 2012. Dostupné z URL: <http://cs.wikipedia.org/wiki/Internet_Message_Access_Protocol>

[9] *Wikipedie - Otevřená encyklopedie* SMTP [online]. 2012. Dostupné z URL: <http://cs.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol>

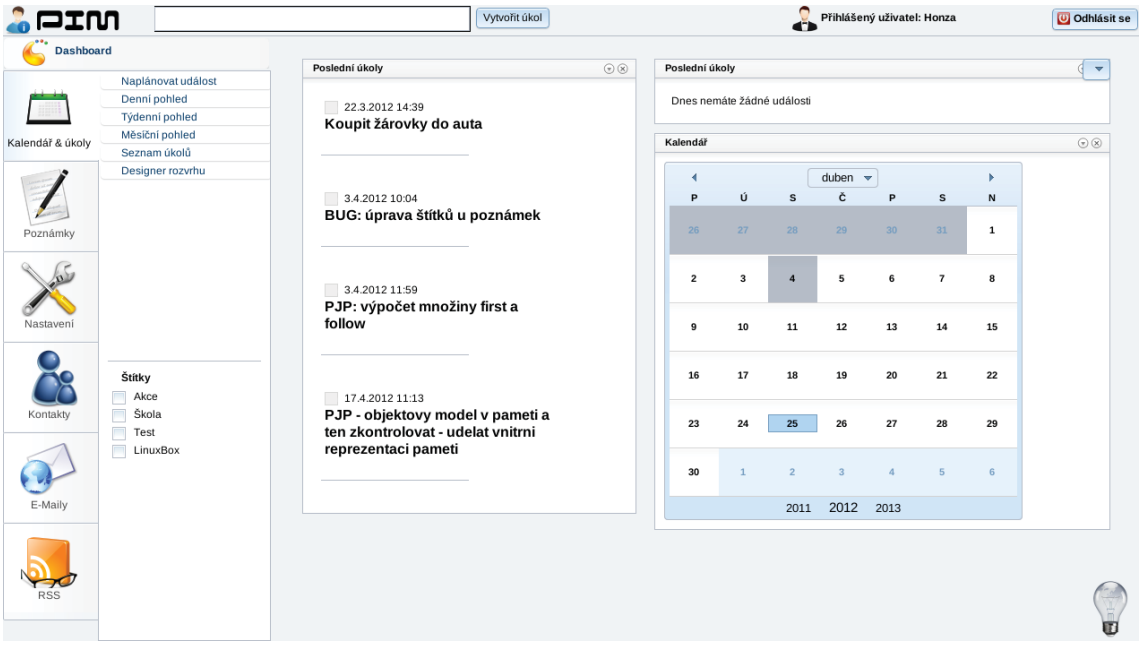
[10] Russell, Matthew A. *Dojo: The Definitive Guide* Farnham : O'Reilly, 2008.

[11] Crockford, D., *RFC 4627* The application/json Media Type for JavaScript Object Notation (JSON) http://tools.ietf.org/html/rfc4627

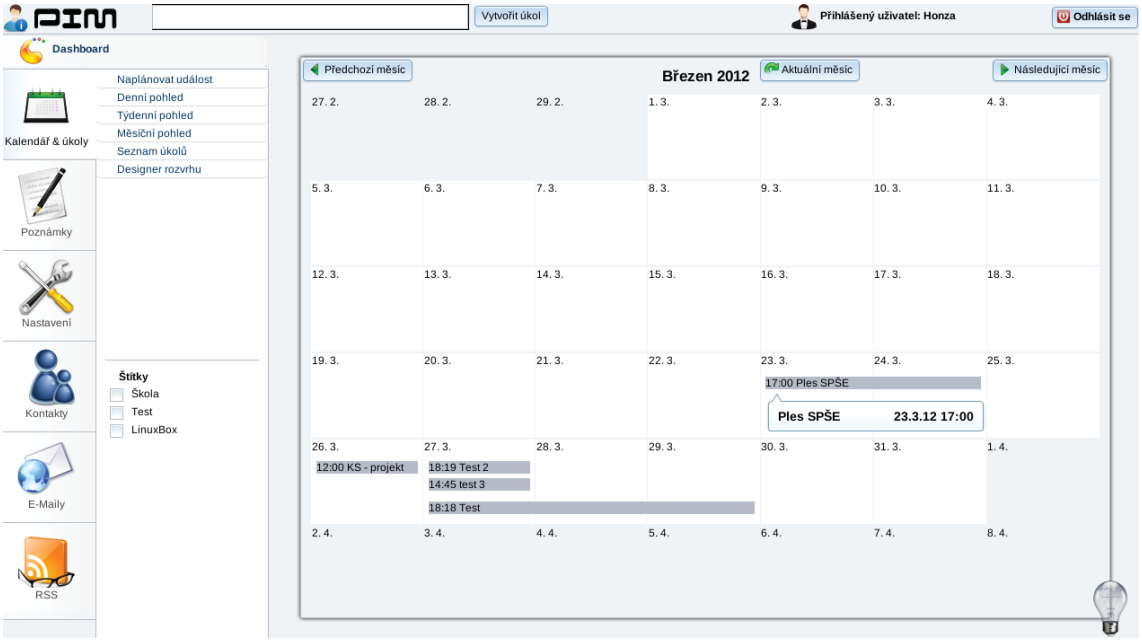
[12] Crispin, M., *RFC 3501* INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1 University of Washington 2003

[13] Klensin, J. *RFC 2821* Simple Mail Transfer Protocol AT&T Laboratories 2001

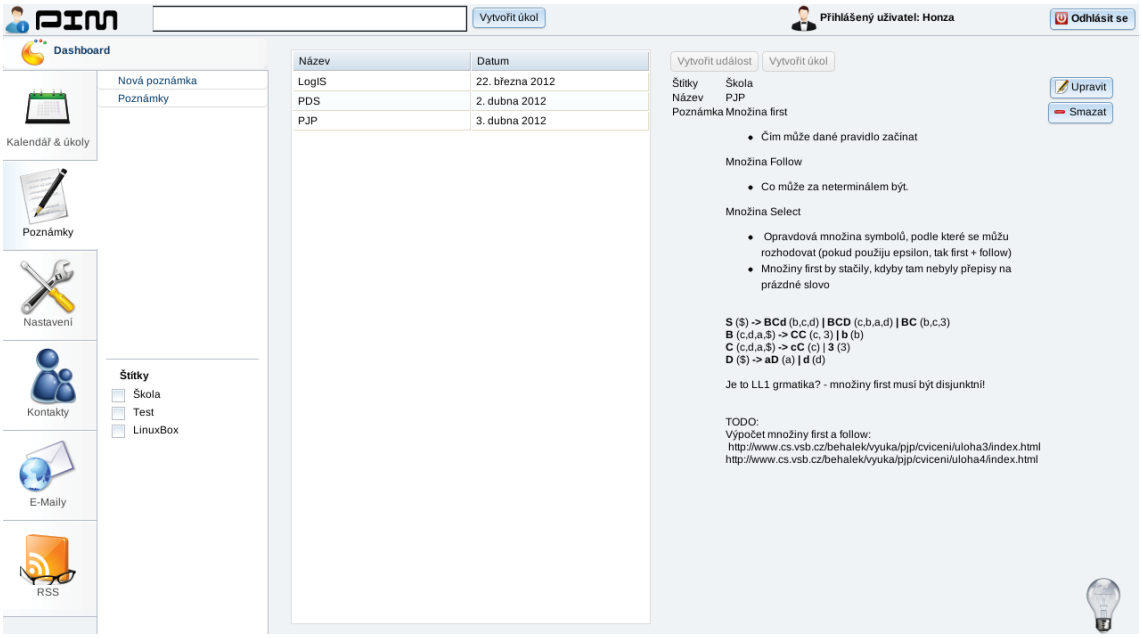
A Obrázky aplikace



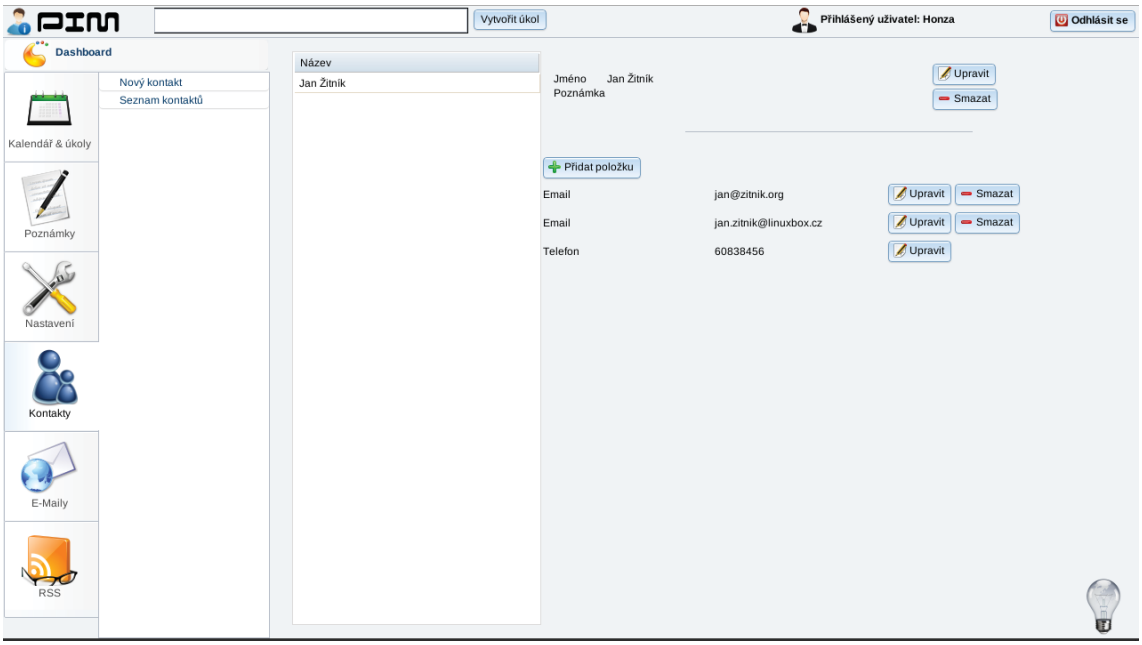
Obrázek 1: Dashboard



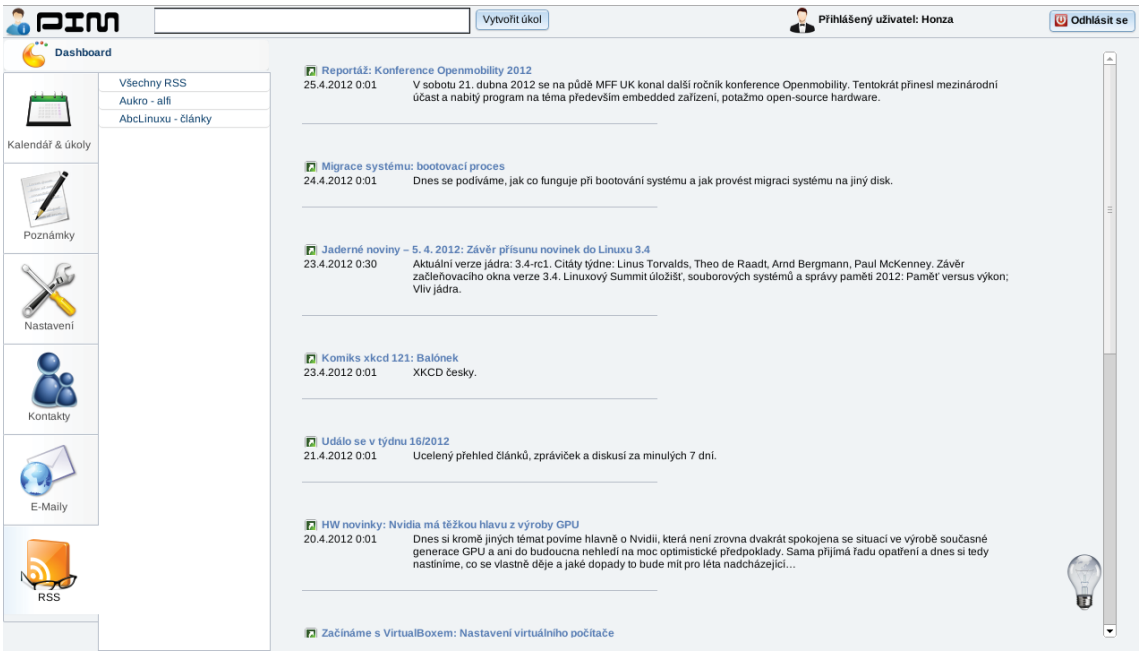
Obrázek 2: Měsíční pohled



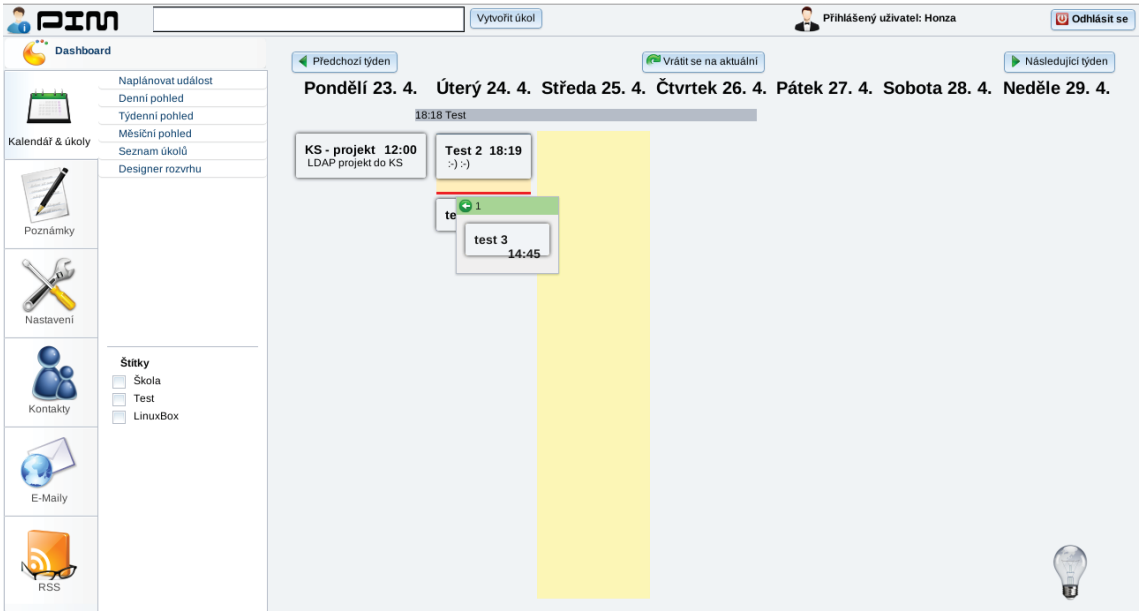
Obrázek 3: Poznámky



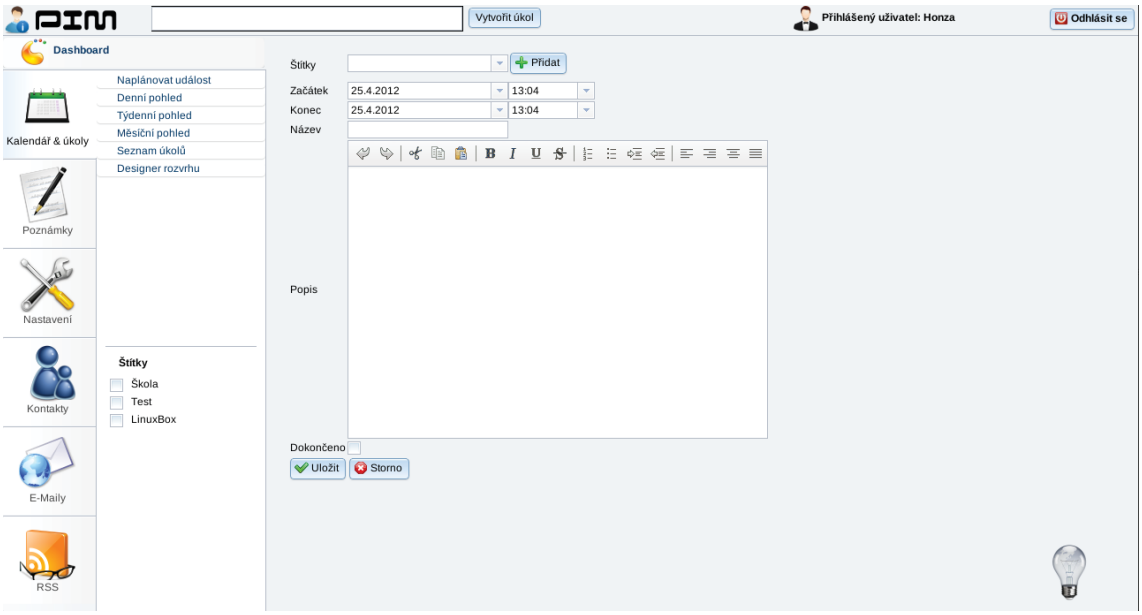
Obrázek 4: Kontakty



Obrázek 5: RSS feedy



Obrázek 6: Týdenní pohled a drag&drop



Obrázek 7: Vytváření nové události